Designing Biological Sequences via Meta-Reinforcement Learning and Bayesian Optimization

Leo Feng **Padideh Nouri** **Aneri Muni** **Yoshua Bengio** **Pierre-Luc Bacon**

Mila – Université de Montréal

Abstract

Designing functionally interesting biological sequences pose challenges due to the combinatorially large space of the problem. As such, the acceleration of exploration through this landscape can have a substantial impact on the progress of the medical field. Motivated by this, we propose MetaRLBO where we (1) train an autoregressive generative model via Meta-Reinforcement Learning augmented with surrogate reward functions and exploration bonus to navigate through the sequence space efficiently. The Meta-RL policy is trained over a distribution of beliefs (i.e., proxy oracles) of the objective function, encouraging the policy to generate diverse sequences. Due to the large-batch and low-round nature of the wet-lab evaluations (true function evaluation), we (2) perform a more targeted evaluation through Bayesian Optimization. Our in-silico experiments show that meta-learning over such ensembles provides robustness against reward misspecification and achieves competitive results compared to existing strong baselines.

1 Introduction

Designing biological sequences has been a long-standing challenge due to the large chemical space with extremely sparse functionally interesting sequences [8]. The time and cost-intensive objective function evaluations such as wet lab experiments required to assess these sequences, merits the need for a principled way to tackle such problems in a black-box fashion. We propose Bayesian Optimization (BO), which approximates the expensive objective function (i.e., *true oracle*) of the optimization problem, with a cheap surrogate model and evaluates samples using an acquisition function to explore (i.e., select uncertain but informative sequences) or exploit (i.e., select sequences with high predicted value).

Although acquisition functions are relatively cheap to evaluate, naively searching over the space of possible sample locations can quickly become intractable. In previous work, Swersky et al. [21] proposed to train a policy such that it modifies a population of sequences to directly optimize the acquisition function. Belanger et al. [4] propose to brute-force enumerate over the search space for shorter sequences and to use regularized evolution [16] for designing longer sequences. Romero et al. [17] propose to construct sequences with enhanced thermostability using crossover on an existing dataset of sequences and selects sequences using GP-UCB [20].

In this work, we propose to treat the problem of biological sequence discovery as training a generator that proposes promising candidate sequences and use Bayesian Optimization as a selection procedure. Generative models give us the ability to jump to the regions of the landscape that were never searched before. In this paper, we present a method called MetaRLBO where we frame the problem of training a generator as a Meta-Reinforcement Learning problem and apply Bayesian Optimization for batch

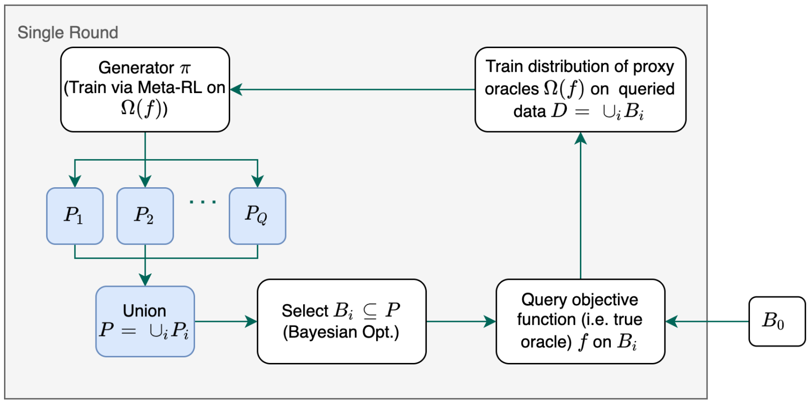

Single Round
Generator π (Train via Meta-RL on $\Omega(f)$)
Train distribution of proxy oracles $\Omega(f)$ on queried data $D = \cup_i B_i$
P_1
P_2
$\cdots$
P_Q
Union $P = \cup_i P_i$
Select $B_i \subseteq P$ (Bayesian Opt.)
Query objective function (i.e. true oracle) f on B_i
B_0

Figure 1: Schematic of the MetaRLBO Framework.

black-box function optimization. In each round, we design our tasks by constructing a distribution of *proxy oracles* and train our generator via Meta-Reinforcement Learning. Then, we generate sequences with this meta-learned generator and finally select the sequences to query the *true oracle* via Bayesian Optimization. Our experiments show that MetaRLBO achieves competitive performance compared to existing strong baselines. In addition, we analyse the uncertainty estimates given by various surrogate models for batch sequence design.

2 Methodology

The optimization problem consists of generating a sequence that maximises the expected reward $s_t^* = \text{argmax}_{s_t \in S} f(s_t)$. The sequence design can be viewed as a Markov decision process (MDP) where s_T is a sequence of length T and of alphabet $\Sigma_{t=1}^{T} a_t$, that maximises an experimentally measured value (*oracle*) $f : S \to \mathbb{R}$. For example in the Anti-Microbial Peptide (AMP) design task, $f(s)$ is defined as the antimicrobial efficacy of the candidate sequence s, which can be computed using wet lab experiments. In particular, we are interested in a specific case of black-box optimization: one which requires few rounds of evaluations over large batches, typical of wet lab experiments over biological assays (DNA, proteins, RNA, etc). This setting is one where in each round r we want to minimize an objective function using the information from previously generated sequences. In each round, our algorithm generates a set of candidate sequences $\mathcal{B}_r = \bigcup_{i=1}^{B} s_i$ where B is the size of the batch. The sequences are then evaluated according to our oracle f and added to the queried data $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s, f(s)) | s \in \mathcal{B}_r\}$ and a new round is started. The objective is to maximise the score of aggregated sequences $\max_{s \in \bigcup_i \mathcal{B}_i} f(s)$.

2.1 Inner-Outer Loop Optimization

We approach this problem from a reinforcement learning perspective. In this framework, our generator iteratively constructs new sequences by appending one element at a time to a string (initially empty) until it chooses to stop. We view this procedure as a sequential decision making problem where the action space is discrete and consists of all the possible symbols of a given alphabet (e.g., amino acids) and the state space $\mathcal{S} = \bigcup_{t=1,\ldots L} \mathcal{A}^t$ is the set of all possible sequence prefixes where L is the length of the sequence. In other words, at time step t, the state is $s_t = a_0, \ldots a_{t-1}$ s.t. $a_i \in \mathcal{A}$. Because the state changes according to the action of the concatenation operation, the transition probability function in this MDP is fully known and deterministic, ie: $P(s_{t+1}|s_t, a_t) \triangleq \mathbb{1}\,(s_{t+1} = \text{concat}(s_t, a_t))$ In the current instantiation of our algorithm, we provide our system with feedback on the generated string only at the end of the sequence construction and not during.

The nature of our problem prevents us from querying the oracle (wet lab) directly to evaluate the reward function. Hence, we devise a hierarchical approach in which a Bayesian optimization procedure orchestrates the interaction with the oracle while using samples from the RL generator in the *inner loop*. Figure 1 provides an overview of the interaction protocol between all components of our system. In this inner loop, the generator is trained over an ensemble of estimated reward functions acting as a proxy to the expensive true oracle. We are therefore under a model-based RL setting, but one where the transition dynamics are fully known and deterministic. Learning an ensemble of reward functions offers many advantages: it provides uncertainty estimates readily usable by the BO procedure and provides more robustness in the presence of error in the estimated reward model. However, in order for this approach to be practical, we need to be able to learn good policies sufficiently fast for each such proxy oracle (reward function). We address this issue using a MAML-like meta-learning approach where we meta-learn initial parameters for the generator.

2.1.1 Inducing a Distribution over Proxy MDPs

We learn a distribution over proxy oracles $\Omega(f)$ by training on the queried dataset $D = \bigcup_i B_i$. The initial dataset B_0 can either be obtained from previous experiments (e.g., related wet-lab experiments) or sampled randomly according to a prior distribution. In order to induce a distribution over possible MDPs, we sample different subsets of the current dataset D, fit a proxy-oracle model to predict the score, and train an RL agent (generator) for each sub-sampled dataset of size $p|D|$ where $0 < p \leq 1$.

2.1.2 Training the generator via Meta-RL

The particular flavor of meta-learning considered in this paper is based on the adaptation procedure of MAML [9]. In each meta-training step, we sample a set of V proxy oracles $\hat{f}_1, \hat{f}_2, \dots \hat{f}_V \in \Omega(f)$. Each proxy oracle specifies a different MDP (i.e. task) M_i for which the reward function is given by $\hat{f}_i$. During the adaptation phase of our generator, we fine-tune π_θ for K steps using a REINFORCE gradient estimator from a learned θ^0. In order to learn the initialisation, we backpropagate through the K gradient steps.

$$\theta^i_{t+1} \leftarrow \theta^i_t - \alpha \nabla_\theta \mathcal{L}(\theta^i_t; M_i) \ ,$$
$$\theta^0_{t+1} \leftarrow \theta^0_t - \gamma \frac{1}{V} \sum_{i=1}^{V} \nabla_\theta \mathcal{L}(\theta^i_K; M_i)$$

While higher-order differentiation of the REINFORCE surrogate loss results in a biased meta-gradient [10], we choose to ignore this technicality in favour of increased computational efficiency as commonly done in practice with other algorithms.

2.1.3 Generating sequences

When generating sequences to evaluate, we sample a number of proxy oracles $\bar{f}_1, \bar{f}_2, \dots, \bar{f}_Q \in \Omega(f)$, each reflecting a belief of the true oracle f. We then finetune each generator for some pre-defined number of steps for each tasks, giving us a set of policies $\pi_{\theta^{(1)}}, \pi_{\theta^{(2)}}, \dots \pi_{\theta^{(Q)}}$. At the end of this procedure, we then query $|P|$ sequences from each policy that we gather into a batch $\mathcal{P} = \bigcup_{i=1,\dots,Q} P_i$. Since the reward function of each MDP is different, generated sequences from different policies $\pi_{\theta^{(1)}}, \pi_{\theta^{(2)}}, \dots \pi_{\theta^{(Q)}}$ provide uncertainty about the generation process as a whole.

Promoting Diversity Within a Round In each round of interaction at the outer level (from BO to wet lab), a batch of candidate sequences to be presented to the Bayesian Optimization must be formed using the ensemble of generators. By virtue of using a different reward function in each MDP, the corresponding learned generators tend to be different from each other, i.e. the learned optimal policies need not be the same. Hence, the combination of subset sampling and meta-learning readily provides a mechanism for generating more diverse candidates in each round. That is: it provides a form of meta-exploration on top of the built-in exploration mechanism of softmax policies.

Promoting Diversity Across Rounds Similar to the existing literature on count-based methods and density models for promoting exploration [5, 2], we augment the reward with an exploration bonus. This method encourages the generation of diverse sequences relative to the ones queried in the previous rounds. This augmented reward is defined as $r = f(s) - \lambda \text{dens}_\epsilon(s)$ where $\text{dens}_\epsilon(s)$ is the weighted number of sequences with a distance less than a threshold ϵ from s. λ is a hyperparameter that controls the the strength of exploration.

2.1.4 Selection via Bayesian Optimization

When selecting sequences, we evaluate the generated sequences P according to an acquisition function. For the surrogate model, we can re-use the proxy models $(\bar{f}_1, \dots \bar{f}_Q)$ sampled from the proxy oracle distribution $\Omega(f)$ that were used for training our policies. Aggregating the proxy models together, we get a surrogate model comprising of an ensemble of neural networks without the need for additional computation. Alternatively, we can train a new surrogate model given the queried dataset $\bigcup_i B_i$. However, in our experiments, we found that re-using the proxy models works well in practice. For evaluation, we greedily select the sequences $B_i \subseteq P$ that maximise the acquisition function and query them using the etobjective function (i.e. true oracle) f.

3 Experiments

For the sake of conciseness, training details are included in the Appendix. We test our approach on three kinds of sequence optimization problems: designing Anti-Microbial Peptide sequences (AMPs), Ribonucleic Acid (RNA) sequences, and sequences that maximise a synthetic Alternating Ising Model. In the AMP problem, the sequences are of variable lengths. In the Alternating Ising and the RNA14 Task, the sequence length is fixed.

Antimicrobial Peptides (AMP) AMPs are small peptides with amino acids as their building blocks and a length generally ranging from 8 to 75 (amino acids). This task consists of an alphabet of size 20 and a maximum length of 50, which gives rise to a search space of size $O(20^{50})$. Following [2], we train a random forest classifier to predict whether a sequence is antimicrobial towards a certain pathogen and we use that as our ground truth (wet lab) simulator. In our experiments, we perform 12 rounds with a batch size of 250.

Alternating Ising Model We consider the synthetic problem of generating a string of alternating characters [21]. The string with the highest score is one that only alternates between two characters. We consider the problem setting with lengths of 20 with an alphabet of size 20. As such, the search space of this problem is $O(20^{20})$. In our experiments, we perform 16 rounds with a batch size of 500.

Ribonucleic Acid (RNA) We design RNA sequences with length 14 from the alphabet of size 4 nucleotides, this would give us a search space of $O(4^{14})$. Here the optimization problem can be defined as finding a sequence that maximises the negative binding energy towards a hidden RNA target of length 50. To simulate the ground truth oracle we use FLEXS package. [19, 14] In our experiments, we perform 12 rounds with a batch size of 100.

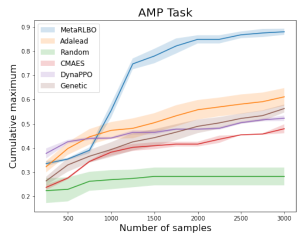
3.1 Baselines

In our experiments, we compare against several baselines including exploration algorithms proposed in FLEXS [19] and Bayesian Optimization methods proposed in [21]. Primarily, we consider: (1) **Random**: A baseline that mutates a random previously measured sequence. (2) **Genetic**: A naive genetic algorithm that uses a Wright-Fisher model and single point mutations and recombinations. [19]. (3) **CMA-ES**: Covariance Matrix Adaptation Evolution Strategies [11, 19]. (4) **DynaPPO**: A model-based RL algorithm that learns a reward function given by the mean predicted value of an ensemble of various different models such as random forests, gaussian processes, and bayesian ridge regression. [2, 19]. (5) **Adalead**: A model-guided evolutionary greedy algorithm [19].

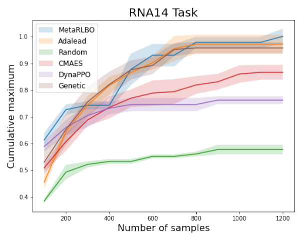
3.2 Results

For a fair comparison, mutative methods are initialised with a random string, to ensure that no prior knowledge is given to the method. Figure 2a shows results using MetaRLBO on the AMP, RNA, and Alternating Ising Model problems. In Table 1, we compare our method with Bayesian Optimization methods and evolutionary methods. We see that MetaRLBO outperforms several existing baselines.

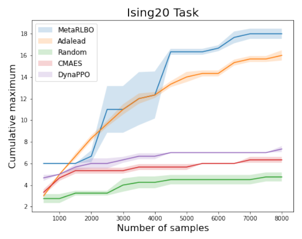
Uncertainty Model Prior work considered using an ensemble of MLPs as the surrogate model for Bayesian Optmization [4, 21] In Figure 2d, we show an uncertainty calibration plot comparing various surrogate models trained on sequences selected in the first n rounds and evaluated on the sequences selected in the $n+1$-th round. This measures how well a surrogate model is at estimating uncertainty in a multi-round setting. Calibration in the regression setting [13] means that y_t should fall in a $c\%$ confidence interval $c\%$ of the time. This means that in Figure 2d, we would want surrogate models to achieve as close to $y = x$ as possible. To generate the data for the analysis, we run MetaRLBO on the AMP task using UCB as our acquisition function and an ensemble of CNNs as our surrogate model.



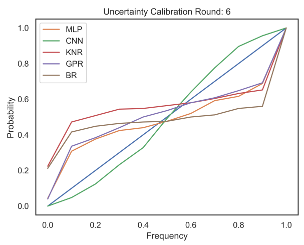
(a) AMP



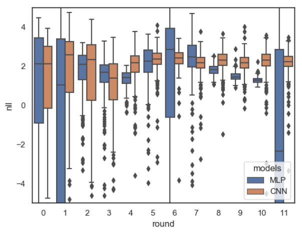
(b) RNA14



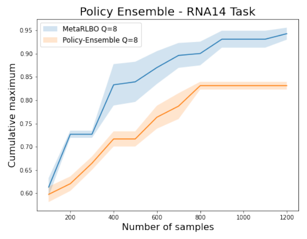
(c) Alternating Ising



(d) Uncertainty Calibration Plot



(e) Box Plot of NLL



(f) Ensemble of Policies

Figure 2: (a), (b), (c): Cumulative max scores given by the objective f. (d), (e), (f): Analysis Plots

Alternating Chain	Length 20	
Acq. Function	UCB	POST
Single Mutant	14.67	14.67
Regularized Evol	14.67	14.67
BO + Single Mutant	15.33	13.67
BO + Regularized Evol	16.67	15.00
BO + DES	16.67	16.33
MetaRLBO	**18.00**	**17.00**

Table 1: Alternating Chain Ising20 task. Previous results obtained from [21]

Proxy Model	Cumul. Max
CNN	18
MLP	15

Table 2: Ising20 Task using UCB. We compare the performance of MetaRLBO when trained on different proxy oracles.

In Figure 2d, we considered surrogate models such as an ensemble of feedforward neural networks with three layers (32, 8, 4 units and $p = 1.0$) used in Belanger et al. [4] and Swersky et al. [21], ensemble of convolutional neural networks ($p = 1.0$), ensemble of Bayesian Ridge Regressors ($p = 0.8$), ensemble of K-Neighbors Regressors ($p = 0.8$), and Gaussian Processes. The results show that an ensemble of CNNs is better calibrated than the other surrogate models. We also measure the Negative Log-likelihood (see Figure 2e), another popular metric for evaluating predictive uncertainty [7]. We see that the ensemble of MLP has higher variance.

We also evaluate the uncertainty models empirically in practice on the Alternating Ising Model task. In this experiment, we run MetaRLBO from scratch using either MLP or CNN as the proxy model $\Omega(f)$. In practice, we found CNNs to perform better (see Table 2). The result of our analysis suggests that convolutional neural networks are better suited for uncertainty estimation in biological sequence design than that of feedforward neural networks.

Ensemble of Policies. As an ablation, we compare MetaRLBO with training an ensemble of policies. Instead of fine-tuning different Q policies from a set of meta-trained parameters, we train Q different policies from scratch per round such that each policy optimises for a different proxy oracle. In these experiments, we set $Q = 8$, generating 256 sequences from each policy. In our MetaRLBO experiments, we additionally set $V = 4$ and $K = 2$. Empirically, we found that training a policy via meta-reinforcement learning improved the performance (see Figure 2f).

Summary. In this work, we proposed MetaRLBO, a black-box optimization method that combines a generator based on Meta-Reinforcement Learning and a selection procedure via Bayesian Optimization. Our results show that MetaRLBO is competitive with other strong baselines. Furthermore, we analysed the uncertainty estimates of surrogate models and find that an ensemble of CNNs is better calibrated than an ensemble of MLPs as used in prior work.

References

[1] Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans. Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning*, pages 151–160. PMLR, 2019.

[2] C. Angermueller, D. Dohan, D. Belanger, R. Deshpande, K. Murphy, and L. Colwell. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HklxbgBKvr`.

[3] J. F. Bard. *Practical Bilevel Optimization*. Springer US, 1998.

[4] D. Belanger, S. Vora, Z. Mariet, R. Deshpande, D. Dohan, C. Angermueller, K. Murphy, O. Chapelle, and L. Colwell. Biological sequences design using batched bayesian optimization. 2019.

[5] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, (Nips):1479–1487, 2016. ISSN 10495258.

[6] Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. 01 2002. doi: 10.1109/IJCNN.1991.155621.

[7] J. Q. Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, B. Schölkopf, and J. Quiñonero Candela. *Evaluating Predictive Uncertainty Challenge*, volume 3944 of *Lecture Notes in Computer Science*, page 1–27. Springer, machine learning challenges - evaluating predictive uncertainty, textual entailment and object recognition systems edition, January 2006. URL `https://www.microsoft.com/en-us/research/publication/evaluating-predictive-uncertainty-challenge/`.

[8] A. Capecchi and J. L. Reymond. Peptides in chemical space. *Medicine in Drug Discovery*, 9:100081, 2021. ISSN 25900986. doi: 10.1016/j.medidd.2021.100081. URL `https://doi.org/10.1016/j.medidd.2021.100081`.

[9] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[10] J. N. Foerster, G. Farquhar, M. Al-Shedivat, T. Rocktäschel, E. P. Xing, and S. Whiteson. Dice: The infinitely differentiable monte carlo estimator. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1524–1533. PMLR, 2018.

[11] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[13] V. Kuleshov, N. Fenner, and S. Ermon. Accurate uncertainties for deep learning using calibrated regression. In *International Conference on Machine Learning*, pages 2796–2804. PMLR, 2018.

[14] R. Lorenz, S. H. Bernhart, C. H. Zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker. ViennaRNA package 2.0. *Algorithms for molecular biology*, 6(1):26, 2011.

[15] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

[16] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[17] P. A. Romero, A. Krause, and F. H. Arnold. Navigating the protein fitness landscape with gaussian processes. *Proceedings of the National Academy of Sciences*, 110(3):E193–E201, 2013. doi: 10.1073/pnas.1215251110. URL `https://www.pnas.org/content/110/3/E193`.

[18] J. Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 1987.

[19] S. Sinai, R. Wang, A. Whatley, S. Slocum, E. Locane, and E. D. Kelsic. Adalead: A simple and robust adaptive greedy search algorithm for sequence design. *arXiv preprint arXiv:2010.02141*, 2020.

[20] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[21] K. Swersky, Y. Rubanova, D. Dohan, and K. Murphy. Amortized bayesian optimization over discrete spaces. In J. Peters and D. Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 769–778. PMLR, 03–06 Aug 2020. URL `https://proceedings.mlr.press/v124/swersky20a.html`.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

[23] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

A Appendix: Glossary

B_0	Initial dataset, either from available wet-lab experiemnts or sampled according to a prior distribution
B_i	Batch of samples selected according to BO for every round i
θ^0	Initialization of policy parameters
f	True oracle (This is a model to approximate wet-lab results and f is trained on B_0 dataset)
$\hat{f}$	Sampled proxy oracle to specify reward
$\bar{f}$	Sampled proxy oracle for sequence generation. $\bar{f}$ chosen based on some BO acquisition function
K	Number of steps for fine-tuning policy π_θ
Q	Number of proxy oracles sampled
P	Number of sequences queried from each policy
$\mathcal{A}$	All possible alphabets to form given type of sequence
L	Length of sequence to be generated

B Background

B.1 Meta-Reinforcement Learning

The specific flavor of Meta-Reinforcement Learning (Meta-RL) considered in this paper is one where we have a distribution of Markov Decision Problems (MDPs) $\Omega(\mathcal{M})$ which we aim to control optimally. Each such MDP $\mathcal{M}_i \in \Omega(\mathcal{M})$ is defined over discrete state and action space $\mathcal{S}$ and $\mathcal{A}$ respectively. In the general case, each MDP can also have its own transition probability function $P_i : S \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and reward function $R_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In our context, the problem structure is such that the transition function is known and shared across MDPs but the reward function varies.

We consider a performance criterion based on the expected discounted return over a finite horizon of length H in searching for an optimal randomized policy $\pi : \mathcal{S} \to \text{Dist}(\mathcal{A})$. We write the expected return $\mathcal{J}$ under π:

$$\mathcal{J}(\pi) = \mathbb{E}_{P_i,\pi}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1})\right] \quad ,$$

γ is a discount factor.

Model Agnostic Meta-Learning (MAML) MAML [9] tackles a specific instance of the meta-learning problem [6, 18] with a gradient-based algorithm inspired by bi-level optimization [3]. More specifically, the MAML formulation considers the problem of learning initialisation parameters θ_0 such that the policy adapts (improves) quickly within a few gradient updates to a new task. More precisely, the meta-RL problem considered in MAML can be concisely formulated as:

$$\begin{aligned}
&\min_{\theta_0} \mathbb{E}_{\mathcal{M}' \sim \Omega(\mathcal{M})}[\mathcal{L}(\theta_K^{\mathcal{M}}; \mathcal{M}')] \\
&\text{s.t. } \theta_{k+1} = \theta_k - \alpha \nabla_\theta \mathcal{L}(\theta_k^{\mathcal{M}}; \mathcal{M}) \\
&\qquad \theta_0^{\mathcal{M}} = \theta \qquad \{\forall \mathcal{M} \in \Omega(\mathcal{M})\} \\
&\qquad \mathcal{L}(\theta_k; \mathcal{M}) = \frac{1}{T}\sum_t G_t \log \pi_{\theta_k}(a_t|s_t)
\end{aligned}$$

where K refers to the number of inner loop updates, T is the number of timesteps (i.e. the horizon), θ refers to the meta-parameters, $\{(s_t, a_t, r_t)\}_{t=0}^{T-1}$ refers to a trajectory generated by the policy, and G_t is the return from timestep t.

B.2 Bayesian Optimization

Bayesian Optimization (BO) aims to maximise a black-box (potentially non-differentiable) objective function in a few evaluations. It does so by building a surrogate model (e.g., a GP [15] or ensemble) of the true (oracle) objective function such that it can be queried at a lower cost. Given the surrogate model, BO then proceeds to efficiently compute the posterior distribution over the function scores and suggest promising candidates according to an acquisition function to be evaluated by the true oracle. Acquisition functions not only take into account the score of a candidate as predicted by the surrogate model but also its uncertainty. As a result, BO methods aim to find a suitable trade-off between exploration (gathering informative data) and exploitation (maximization of the black-box score function).

While our algorithm is compatible with any acquisition function, we found that Upper Confidence Bound (UCB) and Posterior Mean heuristics perform well in practice while being simple to implement. The first strategy, UCB, is defined as $\text{AF}_{\text{UCB}} = \mu(s) + \beta\sigma(s)$ where s is a sample, $\mu(s)$ and $\sigma(s)$ are the mean and standard deviation predicted by the surrogate model and β is a hyperparameter controlling the exploration-exploitation trade-off and with larger values favouring sequences of higher uncertainty. In contrast, the Posterior Mean approach selects only sequences with high predicted scores. As a result, the method does not explicitly try to explore uncertain areas. Posterior Mean can therefore be found as a subcase of UCB for the value of $\beta = 0$.

While Gaussian Processes have been the de facto choice in BO application, their poor scalability in high-dimensions [23] and over large datasets rendered them incompatible with modern deep learning tools. In this work, we use instead an ensemble of convolutional neural networks as the surrogate model. In this case, $\mu(s)$ is given by the mean of the predictions from the ensemble and $\sigma(s)$ is similarly estimated using the standard deviation.

C Training Details

Policy Network We implement the policy as a feedforward neural network that takes as input the flattened one-hot encoding of the generated string, i.e. a vector of dimension $\mathbf{R}^{L\times A}$ where L is the length of the sequence and A is the size of the alphabet. The network output is the logits for the distribution over the next character to be generated. We also use entropy bonus to avoid premature convergence [1]. We apply positional encoding [22] to the one-hot representation of the input string.

Surrogate Model In our experiments, we use an ensemble of CNNs for our proxy models. For each ensemble member, the model is trained for 10 epochs using the ADAM optimizer [12] with a batch size of 50 and mean squared error loss. In practice, we found setting $p = 1.0$ sufficient for training neural networks since diversity can be induced by the different neural network initialisation. However, setting $p < 1.0$ is necessary for evaluating different proxy models (see Section 3.2)

Training During meta-training, we sample $V = 4$ proxy models per meta-updates. When generating sequences, we sample $Q = 32$ proxy oracles and generate $|P| = 64$ sequences from each of $\pi_\theta^{(i)}$. In our experiments, we assume no prior knowledge is given, generating B_0 via a random policy.